Fig. 1A

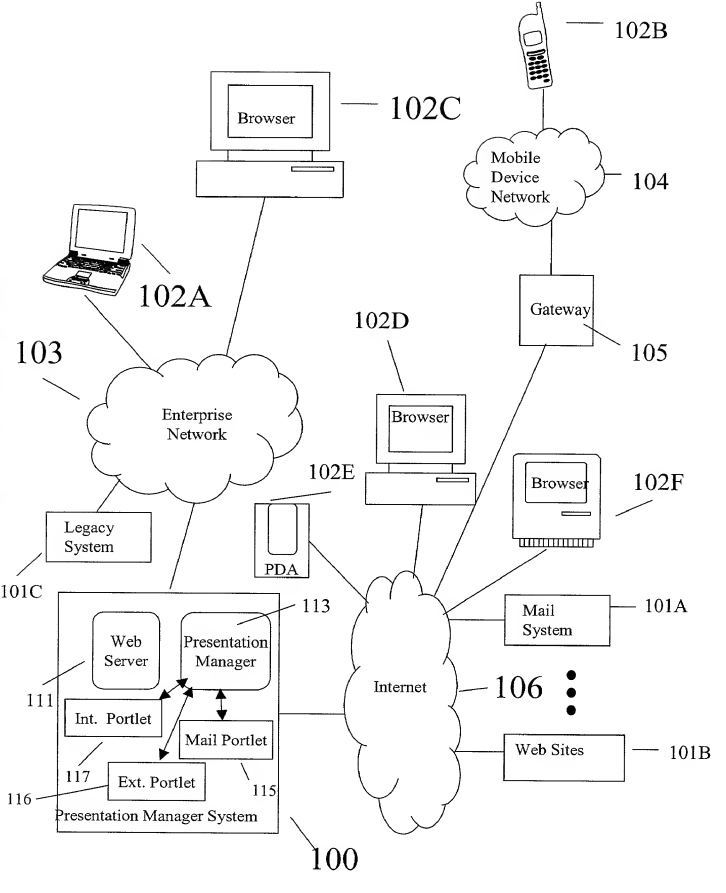

102B
Browser
102C
Mobile Device Network
104
102A
Gateway
105
102D
103
Enterprise Network
Browser
102E
Browser
102F
Legacy System
PDA
101C
113
Mail System
101A
Web Server
Presentation Manager
Internet
106
111
Int. Portlet
Mail Portlet
Web Sites
101B
117
Ext. Portlet
115
116
Presentation Manager System
100

Fig. 1B

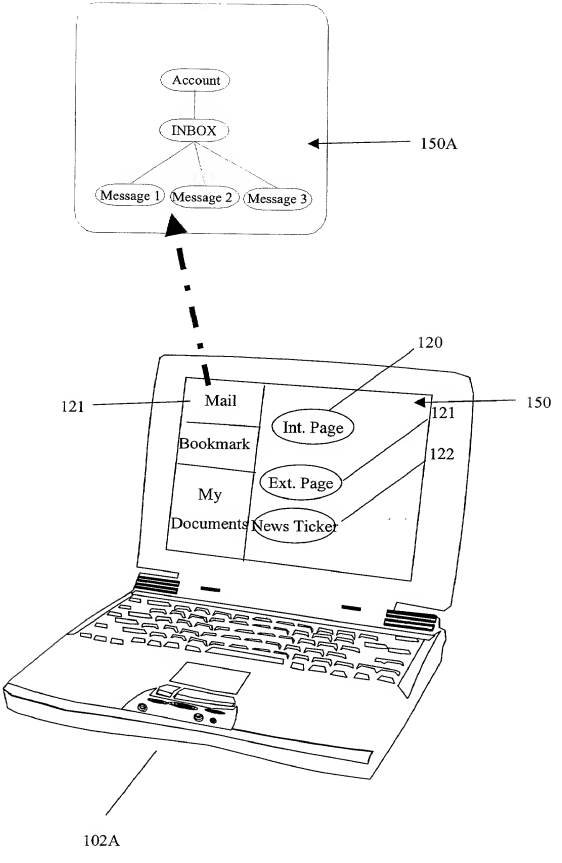

Account
INBOX
Message 1
Message 2
Message 3
150A
120
121
Mail
Bookmark
My Documents
Int. Page
Ext. Page
News Ticker
121
122
150
102A

Fig. 2A

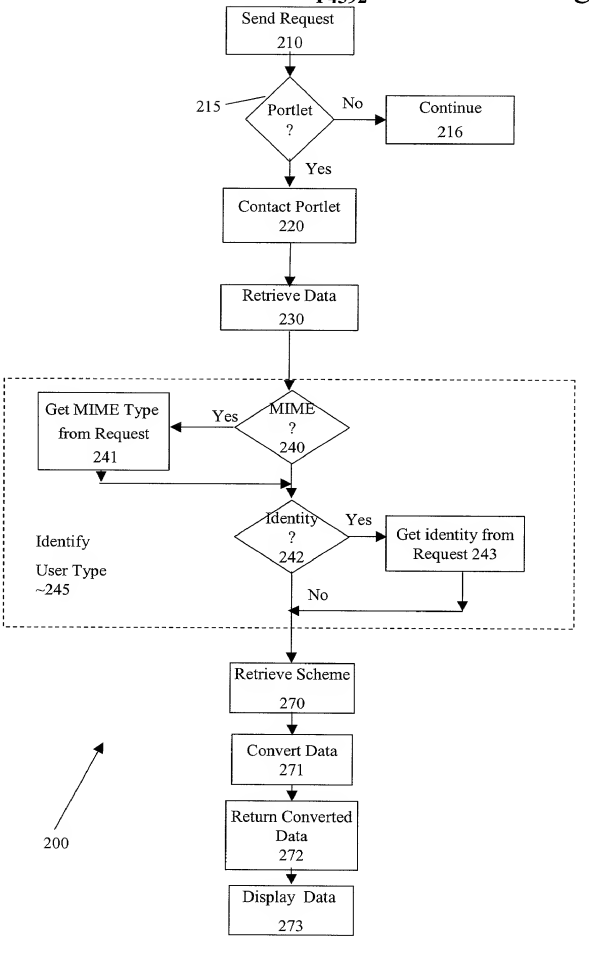
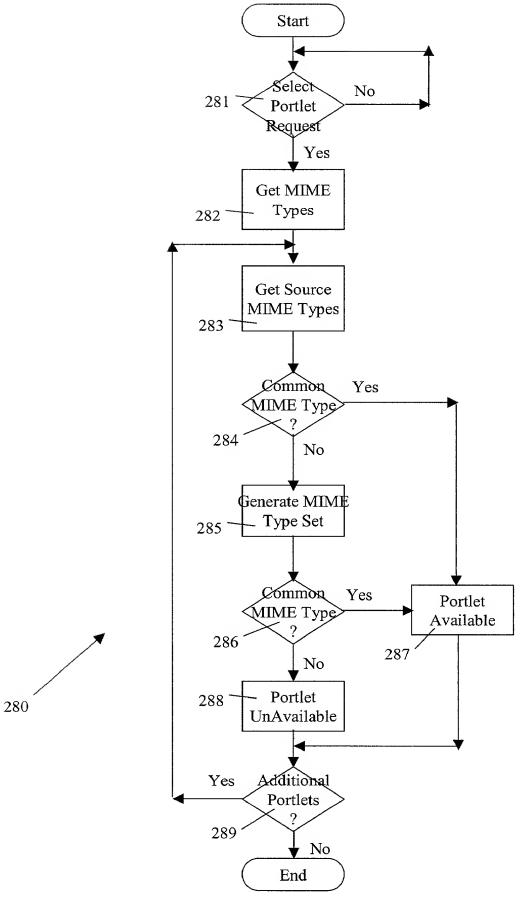

Send Request
210
215
Portlet
?
No
Continue
216
Yes
Contact Portlet
220
Retrieve Data
230
Get MIME Type
from Request
241
Yes
MIME
?
240
Identify
User Type
~245
Identity
?
242
Yes
Get identity from
Request 243
No
Retrieve Scheme
270
Convert Data
271
Return Converted
Data
272
Display Data
273
200

Fig. 2B


Start
Select Portlet Request
281
No
Yes
Get MIME Types
282
Get Source MIME Types
283
Common MIME Type ?
284
Yes
No
Generate MIME Type Set
285
Common MIME Type ?
286
Yes
No
Portlet Available
287
Portlet UnAvailable
288
Additional Portlets ?
289
Yes
No
End
280

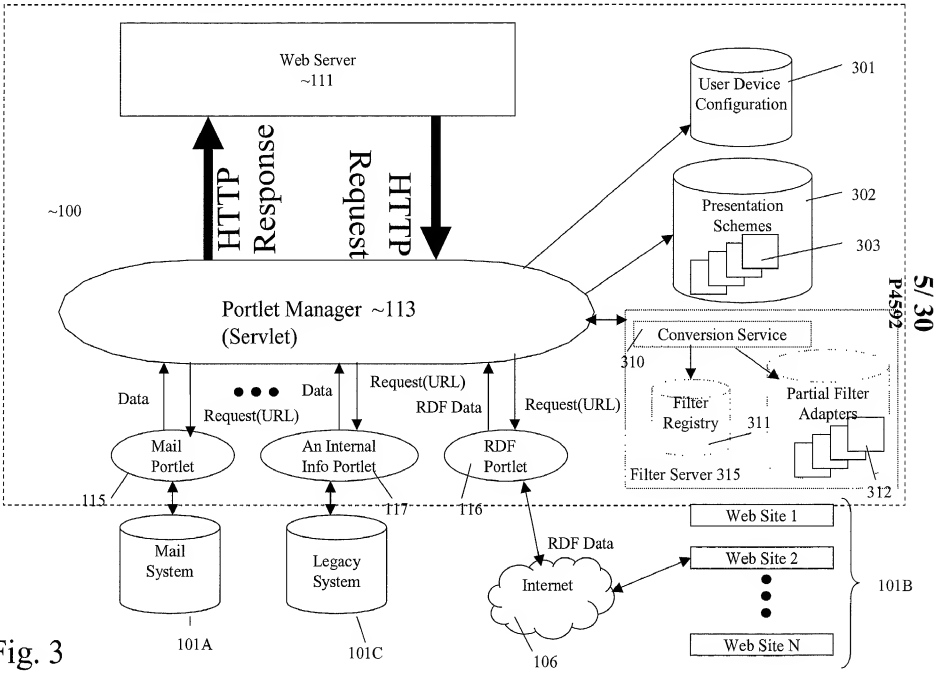
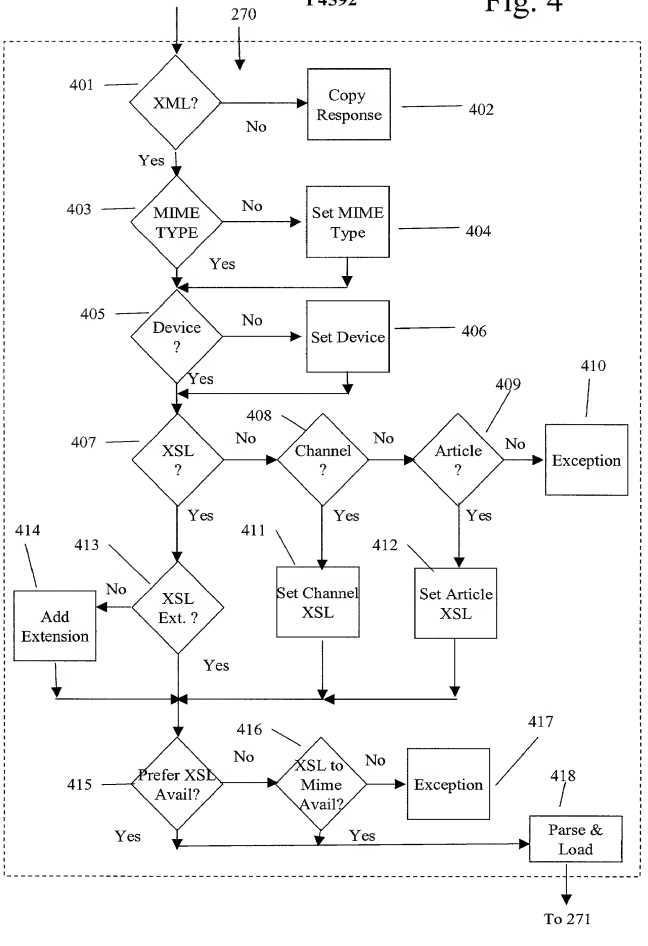

Web Server
~111
HTTP Response
HTTP Request
~100
User Device Configuration
301
Presentation Schemes
302
303
Portlet Manager ~113
(Servlet)
Conversion Service
310
Filter Registry
311
Partial Filter Adapters
312
Filter Server 315
Data
Request(URL)
Data
Request(URL)
RDF Data
Request(URL)
Mail Portlet
An Internal Info Portlet
RDF Portlet
115
117
116
Mail System
Legacy System
RDF Data
Internet
Web Site 1
Web Site 2
Web Site N
101B
101A
101C
106

Fig. 3

Fig. 4


270
401
XML?
No
Copy Response
402
Yes
403
MIME TYPE
No
Set MIME Type
404
Yes
405
Device ?
No
Set Device
406
Yes
410
409
408
407
XSL ?
No
Channel ?
No
Article ?
No
Exception
Yes
Yes
Yes
414
413
411
412
No
XSL Ext. ?
Add Extension
Set Channel XSL
Set Article XSL
Yes
416
417
No
No
415
Prefer XSL Avail?
XSL to Mime Avail?
Exception
418
Yes
Yes
Parse & Load
To 271

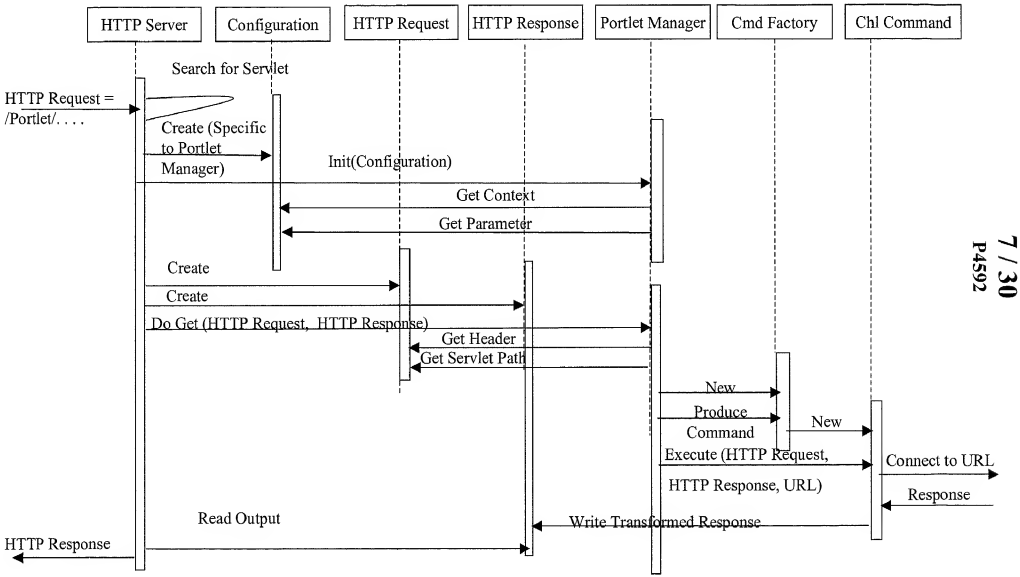

HTTP Server
Configuration
HTTP Request
HTTP Response
Portlet Manager
Cmd Factory
Chl Command
Search for Servlet
HTTP Request =
/Portlet/. . . .
Create (Specific
to Portlet
Manager)
Init(Configuration)
Get Context
Get Parameter
Create
Create
Do Get (HTTP Request, HTTP Response)
Get Header
Get Servlet Path
New
Produce
Command
New
Execute (HTTP Request,
HTTP Response, URL)
Connect to URL
Response
Read Output
Write Transformed Response
HTTP Response

Fig. 5

```
public class PortletManager extends HttpServlet
{

*//This method provides information about a servlet used as portlet manager to a user device. In this example the method returns the name of the servlet and a copyright notice. /*

        public String getServletInfo()
        {
                return "Channel Manager - Copyright 2000 Sun Microsystems, Inc.";
        }
*//
This method is called when the webserver (HTTP server) loads the servlet.  It is only called once for the purpose of initialization.  In this case configuration path names are loaded to the XSL and XML files.
/*
        public void init(ServletConfig configuration)
                throws ServletException
        {
                System.out.println(getServletInfo());
                super.init( configuration );
                m_aServletContext = configuration.getServletContext();
                m_aServletContext = getServletContext();
                String sRootBase      = null;
                String sXslBase       = null;
                String sUserBase      = null;
                String sNoUserName  = null;
                try
                {
                        sRootBase   = new String (getInitParameter("rootbase-uri"));
                        sXslBase    = new String (getInitParameter("xslbase-uri"));
                        sUserBase   = new String (getInitParameter("userbase-uri"));
                        sNoUserName = new String (getInitParameter("nouser-name"));
                }
                catch ( Exception e )
                {
                        System.err.println("some required init parameter is missing");
                        throw new ServletException("init parameter missing");
                }
                . . .
        }
```

FIG. 6

```
/*The doPost method is called every time a HTTP post request arrives at the web server with a request for an URL for which the servlet was configured initially.  Post requests are handled in the same method where get requests are handled, so the doGet() method is called.*/

        public void doPost( HttpServletRequest request,
                            HttpServletResponse response )
                throws ServletException, IOException
        {
                doGet(request, response);
        }

/*The doGet method is called every time a HTTP get request arrives at the webserver with a request for an URL for which the servlet was configured.*/
        public void doGet( HttpServletRequest  aHttpServletRequest,
                           HttpServletResponse aHttpServletResponse )
                throws ServletException, IOException
        {
                try
                {
/*A string is formed which specifies the URL of the servlet based on the information coming with the request.  This string can later be used to specify the servlet as target for a HTTP-request.  This will be done once, because this URL can only change if the configuration of the webserver changes.  In the latter case the servlet would be unloaded and loaded again.*/
                        // do it only once
                        if (m_sPrefixPath.length()==0)
                        {
                                // build prefix path (e.g. www.sun.com:8088/portal/CM/)
                                String  sHostName    = "";
                                String  sContextPath = "";
                                String  sServletPath = "";

                                // get host name part
                                sHostName = aHttpServletRequest.getHeader("Host");

                                if ((sHostName == null) || (sHostName.length() == 0))
                                {
                                        sHostName = aHttpServletRequest.getServerName() +                    ":" +
aHttpServletRequest.getServerPort();
                                }

                                sContextPath = getContextPath(aHttpServletRequest, new Object[0]);
```

FIG. 7A

```
                    sServletPath = aHttpServletRequest.getServletPath();// get servlet path part
                    m_sPrefixPath = "http://" + sHostName + sContextPath;
                    m_aDataFacade.setContextPath(m_sPrefixPath);
                    m_sPrefixPath += sServletPath;                    // concat
                    m_aDataFacade.setPrefixPath(m_sPrefixPath);       // set path on DataFacade

                    m_aCommandFactory = new CommandFactory();         // new command factory
                }
/**The actual work for resolving the request is done by separate objects. For each command specified in the URL given in the request a new
object is instanced which is responsible to resolve the request. This approach is based on a design pattern called the "Command"-pattern. In the
current realization there are the commands „load", „save", „article" and „channel". The object associated with each command is instantiated by
class CommandFactory.
*/
                    // get command pattern and execute
                    Command aCommand =
                        m_aCommandFactory.produceCommand(aHttpServletRequest);
                    aCommand.execute(aHttpServletRequest, aHttpServletResponse, m_aDataFacade);
                }
                catch (Exception e)
                {
                    e.printStackTrace(System.out);
                }
        }

        ...

        private ChannelHook         m_aChannelHook          = null;
        private CommandFactory      m_aCommandFactory= null;
        private DataFacade          m_aDataFacade           = null;
        private ServletContext      m_aServletContext   = null;
        private String              m_sPrefixPath       = "";
}
```


FIG. 7B

```
/*The class CommandFactory parses the head of the specified URL for the request for a command, e.g. „channel“. If a known command is
recognized, the associated command object is created.*/

public class CommandFactory
{
        public Command produceCommand(ServletRequest aRequest)
                throws ServletException
        {
                Command aCommand = null;
                String  sUrl     = null;
//The following method makes sure that this is an HTTP request and gets the URL out of the request.
                if(aRequest instanceof HttpServletRequest)
                {
                        HttpServletRequest sHttpServletRequest = (HttpServletRequest) aRequest;
                        sUrl = sHttpServletRequest.getPathInfo();
                }
                else
                {
                        throw new ServletException( "Request is not of type HttpServletRequest" );
                }
/*The following method tries to recognize the recent command by comparing the head of the URL with the known set of commands.  If a kno
command is found, the associated class is instanced and this instance is returned to the caller.*/
                // careful: sort order of if clauses is important
                if( sUrl != null )
                {
                        if( sUrl.startsWith(CHANNEL) )
                        {
                                aCommand = new ChannelCommand();
                        }
                        if( sUrl.startsWith(ARTICLE) )
                        {
                                aCommand = new ChannelCommand();
                        }
                        else if( sUrl.startsWith(LOAD) )
                        {
                                aCommand = new LoadCommand();
                        }
                        else if( sUrl.startsWith(SAVE) )
```

Fig. 8A

```
                {
                        aCommand = new SaveCommand();
                        System.out.println("- new SaveCommand");
                }
        }
        if( aCommand == null )
        {
                throw new ServletException( "Factory cant produce command for URL" + sUrl );
        }

        return aCommand;
    }

    public final static String LOAD    = "/load";
    public final static String CHANNEL = "/channel";
    public final static String ARTICLE = "/article";
    public final static String SAVE    = "/save";
}
```


Fig. 8B

```
public class LoadCommand extends Command
{
public void execute(ServletRequest  aRequest, ServletResponse aResponse,  DataFacade aDataFacade)
        throws ServletException
        {
                try
                {
                        String          sSessionId                      = getSessionId(aRequest);
                        String          sURL                            = aRequest.getParameter("href");
                        String          sPreferredMimeType              = aRequest.getParameter("Mime");
                        String          sDevice                         = aRequest.getParameter("Device");
                        String          sXSL                            = aRequest.getParameter("XSL");
                        String          sMatch                          = aRequest.getParameter("match");
                        String          sPath                           = null;
                        String          sKey                            = null;
                        String          sValue                          = null;
                        Enumeration     aParameterList                  = aRequest.getParameterNames();
                        Hashtable       aXSLParameters                  = new Hashtable();
                        XmlDocument aFilteredDoc                        = null;
                        boolean         bFound                          = false;
                        if ( sMatch == null || sMatch.equals("") )
                        {
/* this should only be done if there is ONLY one other parameter other than Mime, Device or XSL because the order in the Enumeration is NOT
deterministic and does not equal the order of parameters in the request */
                                while (aParameterList.hasMoreElements() && (!bFound) )
                                {
                                        sKey = (String) (aParameterList.nextElement());
                                                if ((!sKey.equals("Mime"))    &&
                                                        (!sKey.equals("Device")) &&
                                                        (!sKey.equals("XSL"))  )
                                        {
                                                bFound = true;
                                                sValue = aRequest.getParameter(sKey);
                                        }
                                }
                        }
                        else
                        {
/* if there are more parameters the user should set a match parameter to select the key to be used */
```

Fig. 9A

```
                sKey    = sMatch;
                sValue = aRequest.getParameter(sKey);
                bFound          = true;
        }
        if ( !bFound || (sKey.equals("href") && (sValue==null||sValue.equals("root"))) )
        {
                sKey="href";
                sValue=(new URL(aDataFacade.getRootBase())).toString();
        }
        // todo: better
        if              (sKey.equals("href"))           {sPath = "Portlet/@";}
        else if (sKey.equals("subscribed")) {sPath = "Portlet/@";}
        else if (sKey.equals("class"))          {sPath = "Portlet/@";}
        else if (sKey.equals("context"))        {sPath = "PortletUserDevice/@";}
        else if (sKey.equals("sticky"))         {sPath = "PortletUserDevice/@";}
        else if (sKey.equals("floating"))       {sPath = "PortletUserDevice/@";}
        else if (sKey.equals("rollup"))         {sPath = "PortletUserDevice/@";}
        else if (sKey.equals("minimized"))      {sPath = "PortletUserDevice/@";}
        else if (sKey.equals("maximized"))      {sPath = "PortletUserDevice/@";}
        else if (sKey.equals("sizeable"))       {sPath = "PortletUserDevice/@";}
        else                                    {sPath = "PortletUserDevice/parameter";}
        if (sPath!=null)
        {
                aFilteredDoc = aDataFacade.getFilteredDoc(sSessionId, sPath, sKey, sValue);
        }
        XMLResolver aXMLResolver = new XMLResolver(true);
aXMLResolver.registerCatalogEntries("com.sun.star.portal.portlet.manager.dtd");
        if (sXSL.equals("none"))
        {
                String sSystemId = "PortletsAll.dtd";
                String sPublicId = aXMLResolver.getPublicId(sSystemId);
                sSystemId = aDataFacade.getPrefixPath()+"/dtd/PortletsAll.dtd";
                aFilteredDoc.setDoctype(sPublicId,  // public identifier
                                                        sSystemId,      // system identifier
                                                                null);          // no internal subset
                // write it
                if (aResponse instanceof HttpServletResponse)
                {
((HttpServletResponse)aResponse).setHeader("Cache-Control", "no-cache");//HTTP1.1
```

Fig. 9B

```
((HttpServletResponse)aResponse).setHeader("Pragma", "no-cache");       //HTTP1.0
                }
                OutputStream aOutStream = aResponse.getOutputStream();
                aResponse.setContentType(sPreferredMimeType);
                aFilteredDoc.write(aOutStream);
                aOutStream.flush();
                return;
        }
        // build path for xsl file which generates response
        if ( (sPreferredMimeType==null) || (sPreferredMimeType.length()==0) )
        {
                // no mime parameter set so get it via portal servlet
                sPreferredMimeType = aDataFacade.getPreferredMimeType(sSessionId);
        }
        if ( (sDevice==null) || (sDevice.length()==0) )
        {
                // no device parameter set so get it via portal servlet
                sDevice = aDataFacade.getDeviceName(sSessionId);
        }
        if ( (sXSL==null) || (sXSL.length()==0) )
        {
                // no XSL parameter so set direct
                sXSL = "PortletsAll.xsl";
        }
        else
        {
                // if no .xsl extension, add one
                if (sXSL.indexOf(".")==-1)
                {
                        sXSL = sXSL+".xsl";
                }
        }
        // build first chance path to xsl file
        URL     aTempURL        = new URL(aDataFacade.getXSLBase());
        String  sXSLBase        = aTempURL.getFile();
        String  sSubMime        = sPreferredMimeType.substring(sPreferredMimeType.indexOf("/")+1);
        File    aXSLMime        = new File(sXSLBase, sSubMime);
        File    aXSLDevice      = new File(aXSLMime, sDevice);
        File    aXSLFull        = new File(aXSLDevice, sXSL);
```

Fig. 9C

```
        if (!aXSLFull.exists())
        {
                // first chance xsl file not there build second chance (no device specific transformation)
                aXSLFull = new File(aXSLMime, sXSL);

                if (!aXSLFull.exists())
                {
                        // no xsl found -> error
                        throw new ServletException("+++ xsl file not found!!!");
                }
        }
        aXSLParameters.put("TransformURL", aXSLFull.toURL().toString());
        // build parse environment
        ValidatingParser        aValParser              = new ValidatingParser(false);
        XmlDocumentBuilder      aDocBuilder             = new XmlDocumentBuilder();
        XSLTransformEngine      aTransformer    = new XSLTransformEngine();
        XmlDocument                     aSourceDoc              = new XmlDocument();
        XmlDocument                     aTransformerDoc         = new XmlDocument();
        XmlDocument                     aDrainDoc               = new XmlDocument();
        // get/config parser and builder
//      aValParser.setDocumentHandler(aDocBuilder);
        aDocBuilder.setParser(aValParser);
        aValParser.setEntityResolver(aXMLResolver);
aDocBuilder.setDisableNamespaces (false);
        // parse xsl file
        InputSource aTransInputSource = new InputSource(aXSLFull.toURL().toString());
        aValParser.parse(aTransInputSource);
        aTransformerDoc = aDocBuilder.getDocument();
        aTransformerDoc = aDataFacade.setParameters(aTransformerDoc, aRequest, aXSLParameters);
        // transform
aTransformer.createTransform(aTransformerDoc)
                        .transform(aFilteredDoc,aDrainDoc);
        TreeWalker      aTreeWalker             = new TreeWalker(aTransformerDoc.getDocumentElement());
        Element         aWalkerElement          = null;
        String          sPublicId               = "";
        String          sSystemId               = "";
        // set doctype (due to error in XT, which doesnt care for xsl:output if DOM tree is emitted)
        while ( (sPublicId.equals("")) && ((aWalkerElement = aTreeWalker.getNextElement("xsl:output")) != null) )
        {
```

Fig. 9D

```
                sPublicId = aWalkerElement.getAttribute("doctype-public");
            }
            aTreeWalker.reset();

            while ( (sSystemId.equals("")) && ((aWalkerElement = aTreeWalker.getNextElement("xsl:output")) != null) )
            {
                sSystemId = aWalkerElement.getAttribute("doctype-system");
            }
            aDrainDoc.setDoctype(       sPublicId,      // public identifier
                                        sSystemId,      // system identifier
                                        null);          // no internal subset
            // write it
            if (aResponse instanceof HttpServletResponse)
            {
                ((HttpServletResponse)aResponse).setHeader("Cache-Control", "no-cache");        //HTTP1.1
                ((HttpServletResponse)aResponse).setHeader("Pragma", "no-cache");               //HTTP1.0
            }
            OutputStream aOutStream = aResponse.getOutputStream();
            aResponse.setContentType(sPreferredMimeType);
            aDrainDoc.write(aOutStream);
            aOutStream.flush();
        }
        catch (TransformException e)
        {
            e.printStackTrace();
            throw new ServletException("LoadCommand - TransformException");
        }
        catch (SAXException e)
        {
            e.printStackTrace(System.out);
            throw new ServletException("LoadCommand - SAXException");
        }
        catch (IOException e)
        {
            e.printStackTrace(System.out);
            throw new ServletException("LoadCommand - IOException");
        }
    }
}
```

Fig. 9E

```
public class SaveCommand extends Command
{
    public void execute(ServletRequest  aRequest,
                        ServletResponse aResponse,
                        DataFacade      aDataFacade)
    throws ServletException
    {
        String      sSessionId      = getSessionId(aRequest);
        String      sURL            = aRequest.getParameter("href");
        Enumeration aParameterList  = aRequest.getParameterNames();
        while (aParameterList.hasMoreElements())
        {
            String sPath  = null;
            String sKey   = (String) (aParameterList.nextElement());
            if (!sKey.equals("href"))
            {
                String sValue  = aRequest.getParameter(sKey);
                // todo: better
                if          (sKey.equals("title"))          {sPath = "Portlet/";}
                else if  (sKey.equals("login"))             {sPath = "Portlet/";}
                else if (sKey.equals("password"))           {sPath = "Portlet/";}
                else if (sKey.equals("subscribed"))         {sPath = "Portlet/@";}
                else if (sKey.equals("upper-left-x"))       {sPath = "PortletUserDevice/";}
                else if (sKey.equals("upper-left-y"))       {sPath = "PortletUserDevice/";}
                else if (sKey.equals("width"))              {sPath = "PortletUserDevice/";}
                else if (sKey.equals("height"))             {sPath = "PortletUserDevice/";}
                else if (sKey.equals("context"))            {sPath = "PortletUserDevice/@";}
                else if (sKey.equals("sticky"))             {sPath = "PortletUserDevice/@";}
                else if (sKey.equals("floating"))           {sPath = "PortletUserDevice/@";}
                else if (sKey.equals("rollup"))             {sPath = "PortletUserDevice/@";}
                else if (sKey.equals("minimized"))          {sPath = "PortletUserDevice/@";}
                else if (sKey.equals("maximized"))          {sPath = "PortletUserDevice/@";}
                else if (sKey.equals("sizeable"))           {sPath = "PortletUserDevice/@";}
                else                                {sPath = "PortletUserDevice/parameter";}    //todo: rollback
                if (sPath!=null)
                {
                    aDataFacade.setValue(sSessionId, sURL, sPath, sKey, sValue);
                }
            }
```

Fig. 10A

```
        }
        // every request needs a response, so I write "OK" as answer for a save request
        try
        {
            HttpServletResponse aHttpResponse = (HttpServletResponse)aResponse;
            PrintWriter         aPrintWriter  = aResponse.getWriter();
            String              sAnswer       = new String("<HTML>OK</HTML>");
            aHttpResponse.setContentType("text/html");
            aHttpResponse.setContentLength(sAnswer.length());
            aHttpResponse.setDateHeader("Last-Modified", System.currentTimeMillis());
            aHttpResponse.setHeader("Cache-Control", "no-cache");   //HTTP1.1
            aHttpResponse.setHeader("Pragma", "no-cache");          //HTTP1.0
            aPrintWriter.write(sAnswer);
            aPrintWriter.flush();
        }
        catch (IOException e)
        {
            e.printStackTrace();
            throw new ServletException("SaveCommand - IOException");
        }
    }
}
```


Fig. 10B

```
/*The ChannelCommand has to provide the content of a portlet as a response to a request.  The ChannelCommand is derived from the class
Command, which is necessary, because of the use of the „Command"-pattern.
*/

public class ChannelCommand extends Command
{
    public void execute( ServletRequest aRequest,    ServletResponse  aResponse,    DataFacade aDataFacade)
        throws ServletException
    {
        InputStream   aDocInputStream     = null;
        InputSource   aDocInputSource     = null;
        InputSource   aTransInputSource   = null;
        URLConnection     aURLConnection              = null;
        String        sSessionId          = getSessionId(aRequest);
        String        sEncodedURL         = aRequest.getParameter("href");
        String        sPreferredMimeType  = aRequest.getParameter("Mime");
        String        sDevice                     = aRequest.getParameter("Device");
        String        sXSL                        = aRequest.getParameter("XSL");
        String        sDecodedURL         = "";
        String        sResponseContent    = "";
        Hashtable         aXSLParameters              = new Hashtable();
        try
        {
/*If no portlet is specificed in the URL information about all available portlets is returned.  This will be decided by checking the URL for any
additional information.  The information about all available portlets is received from a data container called DataFacade.  This concept is based
on the design pattern Façades.
*/
            //set URL for root channel list
            if ((sEncodedURL==null)    ||
               (sEncodedURL.equals(""))||
               (sEncodedURL.equals("root")) )
            {
                sDecodedURL = aDataFacade.getRootBase();
            }
            else
            {
                sDecodedURL = URLDecoder.decode(sEncodedURL);
            }
```

Fig. 11A

```
/*The part in the URL which specifies the Portlets is an URL which can be used to retrieve the content of the Portlet.  This will be used to build
a corresponding URL object, which will be used later on to build an URLconnection.
*/
                    //get URL connection
                    URL aURL = new URL(sDecodedURL);
                    aURLConnection = aURL.openConnection();
                    aURLConnection.setUseCaches(false);
                    aXSLParameters.put("ProviderURL", aURL.toString());
/*In the following method, all additional fields which are specified in the request, are copied to the new request.  These fields can include, for
example, the name of the browser used on the client side, that means in the user device.
*/
                    // tunnel properties
                    java.util.Enumeration aHeaderNameList =  ((HttpServletRequest) aRequest).getHeaderNames();
                    while (aHeaderNameList.hasMoreElements())
                    {
                        String sKey    = (String) aHeaderNameList.nextElement();
                        String sValue  = ((HttpServletRequest) aRequest).getHeader(sKey);
                        aURLConnection.setRequestProperty(sKey, sValue);
                    }
/*The additional MIME types, which are supported by the portlet manager, are added to a list of accepted MIME types and this list will be added
to the new request.
*/
                    // set own mime types, because some devices accept some  mime types which make them crash...
                    String sSupportedMimeTypes = aDataFacade.getSupportedMimeTypes(sSessionId);
                    aURLConnection.setRequestProperty("accept", sSupportedMimeTypes);
/*The provider of the portlet content is connected and the respective content is read out by using an InputStream
*/
                    // get connection to URL
                    aURLConnection.connect();
                    InputStream aProviderResponse = aURLConnection.getInputStream();
/*In case of a HTTP connection the information in the HTTP header can be used to recognize the MIME type of the response.  If the HTTP
protocol is not used, the extension of the specified URL is checked.
*/
                    if ( aURL.getProtocol().equals("http") )
                    {
                        // get content type
                        sResponseContent = aURLConnection.getContentType();
                    }
                    else
```

Fig. 11B

```
{
/*If it is recognized that a XML file is requested, it is tried to guess which kind of XML file it is by analyzing the DOCTYPE field in the header
of the XML file.  In the case that an XML file based on a certain DTD is recognized, the response type is set to the associated name.
*/
                    // protocol is not http, so get content via file
                    if (sDecodedURL.endsWith(".xml"))
                    {
                        //seems to be an xml file, so get doctype content
                        aProviderResponse.mark(1024);
                        byte[] aBuffer = new byte[1000];
                        aProviderResponse.read(aBuffer);
                        aProviderResponse.reset();
                        String aSearchString = new String(aBuffer);
                        int nDoctypePosition = aSearchString.indexOf("<!DOCTYPE ");
                        if (nDoctypePosition==-1)
                        {
                            //unknown xml, send as binary
                            sResponseContent = "";
                        }
                        else
                        {
                            // found "doctype", determine doctype
                            String aTokenizerFeed = aSearchString.substring(nDoctypePosition);
                            StringTokenizer aTokenizer =  new StringTokenizer(aTokenizerFeed, " ");
                            // get doctype
                            aTokenizer.nextToken();
                            String sResult = aTokenizer.nextToken();
                            if (sResult.equals("Channels"))
                            {
                                sResponseContent ="text/vnd.sun.xml.Channels";
                            }
                            else if (sResult.equals("Article"))
                            {
                                sResponseContent = "text/vnd.sun.xml.Article";
                            }
                            else
                            {
                                //unknown xml, send as binary
                                sResponseContent = "";
```

Fig. 11C

```
                    }
                }
            }
            else
            {
/*If it is not a XML file, the class URLconnection guesses the content type of the response stream.
*/
                //no xml file, so start guessing
                sResponseContent = aURLConnection.guessContentTypeFromStream(
                                        aProviderResponse);
                if (sResponseContent==null)
                {
                    //okay, now I'm helpless
                    sResponseContent="";
                }
            }
        }

        aXSLParameters.put("ProviderMimeType", sResponseContent);

        // xml response?
        if (!sResponseContent.startsWith("text/vnd.sun.xml."))
        {
/*If the response content is not in a XML format handled by the portlet manager, the response is returned as response to the original request
without any modification.
*/
            // upps, no xml, so simply tunnel through as raw data
            aResponse.setContentType(sResponseContent);
            if (aResponse instanceof HttpServletResponse)
            {
                ((HttpServletResponse) aResponse).setDateHeader("Last-Modified", System.currentTimeMillis());
            }
/*The input stream of the response is copied to the output stream of the response of the original request.
*/
            InputStream  aURLConnectionInStream = aProviderResponse;
            OutputStream aResponseOutStream     =
    aResponse.getOutputStream();

            byte[] aBuffer      = new byte[1024];
```

Fig. 11D

```
                int   iReadLength   = 0;
                int   iReadLengthSum = 0;

                while ( (iReadLength=
                        (aURLConnectionInStream.read(aBuffer)))!=-1 )
                {
                        aResponseOutStream.write(aBuffer,0,iReadLength);

                        iReadLengthSum += iReadLength;
                }
                // unfortunately available() not possible,  so I have to use a buffer and set filelength at last
                aResponse.setContentLength(iReadLengthSum);
                aResponseOutStream.flush();
                return;
        }
        else
        {
/*If no preferred MIME type, device type or XSL-stylesheet specified with the request exists, the associated default type is used.
*/
                // yes, so build path for xsl file which generates response
                if ( (sPreferredMimeType==null) || (sPreferredMimeType.length()==0) )
                {
                        // no mime parameter set so get it via portal servlet
                        sPreferredMimeType =  aDataFacade.getPreferredMimeType(sSessionId);
                }
                if ( (sDevice==null) || (sDevice.length()==0) )
                {
                        // no device parameter set so get it via portal servlet
                        sDevice = aDataFacade.getDeviceName(sSessionId);
                }

                if ( (sXSL==null) || (sXSL.length()==0) )
                {
/*If channels are processed, the channel sytlesheet is used, if articles are processed,  the article stylesheet is used.
*/
                        // no XSL parameter set so get via content
                        String sSubMime = sResponseContent.substring(17);
                        if (sSubMime.equals("Channels"))
                        {
```

Fig. 11E

```
                sXSL = "Channels.xsl";
            }
            else if (sSubMime.equals("Article"))
            {
                sXSL = "Article.xsl";
            }
            else
            {
                throw new ServletException(
                    "+++ no correct xsl type!!!");
            }
        }
        else
        {
/*If a XSL parameter is specified with the request, this stylesheet is used.*/
            // if no .xsl extension, add one
            if (sXSL.indexOf(".")==-1)
            {
                sXSL = sXSL+".xsl";
            }
        }
/*The path to find the XSL stylesheet file is built. First, a device specific file is searched for in a subdirectory with the name of the device. If
this file is not available, the more generic device independent stylesheet is used.
*/
        // build first chance path to xsl file
        URL   aTempURL = new URL(aDataFacade.getXSLBase());
        String sXSLBase = aTempURL.getFile();
        String sSubMime = sPreferredMimeType.substring(
                            sPreferredMimeType.indexOf("/")+1);
        File   aXSLMime  = new File(sXSLBase, sSubMime);
        File   aXSLDevice = new File(aXSLMime, sDevice);
        File   aXSLFull  = new File(aXSLDevice, sXSL);
        if (!aXSLFull.exists())
        {
            // first chance xsl file not there build second chance (no device specific transformation)
            aXSLFull = new File(aXSLMime, sXSL);
            if (!aXSLFull.exists())
            {
                // no xsl found -> error
```

Fig. 11F

```
                    throw new ServletException(
                        "+++ "+aXSLFull.getAbsolutePath()+
                        " file not found!");
                }
            }
//All the objects needed to parse XML-documents and for the XSL transformation are set up.
            // build parse environment
            ValidatingParser      aValParser =  new ValidatingParser(true);
            XmlDocumentBuilderaValDocBuilder =  new XmlDocumentBuilder();
            ValidatingParser      aParser = new ValidatingParser(false);
            XmlDocumentBuilderaDocBuilder =  new XmlDocumentBuilder();
            XSLTransformEngineaTransformer =  new XSLTransformEngine();
            XmlDocument           aSourceDoc    = null;
            XmlDocument           aTransformerDoc = null;
            XmlDocument           aDrainDoc =
                                  new XmlDocument();
            // hook own entity resolver for finding dtds
            XMLResolver aXMLResolver = new XMLResolver(true);
            aXMLResolver.registerCatalogEntries("com.sun.star.portal.channel.dtd");
            // get/config parser and builder
            aValDocBuilder.setParser(aValParser);
            aValParser.setEntityResolver(aXMLResolver);
            aValDocBuilder.setDisableNamespaces(false);

            aDocBuilder.setParser(aParser);
            aParser.setEntityResolver(aXMLResolver);
            aDocBuilder.setDisableNamespaces(false);
/*Parse the received response as an XML document and hold the whole document as a DOM tree in memory.
*/
            // parse received xml validating
            aDocInputSource = new InputSource(aProviderResponse);
            aDocInputSource.setSystemId(sDecodedURL);
            aValParser.parse(aDocInputSource);
            aSourceDoc = aValDocBuilder.getDocument();
//The XSL stylesheet is parsed and loaded into a memory.
            // parse xsl file NON validating
            aTempURL = aXSLFull.toURL();
            aXSLParameters.put("TransformURL", aTempURL.toString());
            aTransInputSource = new InputSource(aTempURL.toString());
```

Fig. 11G

```
aParser.parse(aTransInputSource);
aTransformerDoc = aDocBuilder.getDocument();
/*If the loaded document is a list of channels, this document is modified in a memory to which a user is not subscribed or of which the MIME
type is not accepted for the response.  This is necessary, because such content could not be displayed on a user device.
*/
if (sResponseContent.equals("text/vnd.sun.xml.Channels"))
{
    Element   aRootElement = aSourceDoc.getDocumentElement();
    TreeWalker aRootWalker  =  new TreeWalker(aRootElement);
    Element   aChannelElement = null;
    // walk through channel elements
    while ( (aChannelElement = aRootWalker.getNextElement("Channel")) != null)
    {
        // set subscribed attribute
        String sHref =  aChannelElement.getAttribute("href");
        if (aDataFacade.isSubscribed( sSessionId, sHref))
        {
            aChannelElement.setAttribute("subscribed", "true");
        }
        else
        {
            aChannelElement.setAttribute("subscribed", "false");
        }
        TreeWalker aChannelWalker =  new TreeWalker(aChannelElement);
        Element   aMimeElement  = null;
        boolean   bMimeAccepted  = false;
        // walk through mime-type elements of channel
        while (((aMimeElement = aChannelWalker.getNextElement(
            "mime-type")) != null) && (!bMimeAccepted) )
        {
            Node aNode = aMimeElement.getFirstChild();
        String  sMimeType  = new String(aNode.getNodeValue());
        bMimeAccepted = aDataFacade.isMimeTypeAccepted( sSessionId, sMimeType);
    }
    // set accepted attribute
      if (bMimeAccepted)
      {
        aChannelElement.setAttribute( "accepted", "true");
      }
```

Fig. 11H

```
                else
                {
                    aChannelElement.setAttribute("accepted", "false");
                }
            }
        }
/*If the loaded document is an article, it is checked, if this MIME type is accepted as a response.
*/
        else if(sResponseContent.equals("text/vnd.sun.xml.Article"))
        {
            Element    aRootElement = aSourceDoc.getDocumentElement();
            TreeWalker aRootWalker  = new TreeWalker(aRootElement);
            Element    aItemElement = null;

            // walk through item elements
            while ( (aItemElement = aRootWalker.getNextElement("Item")) != null)
            {
                TreeWalker aItemWalker = new TreeWalker(aItemElement);
                Element  aMimeElement     = null;
                boolean bMimeAccepted     = false;
                // walk through mime-type elements of item
                while (((aMimeElement = aItemWalker.getNextElement(
                                                    "mime-type")) != null) && (!bMimeAccepted) )
                {
                    Node aNode = aMimeElement.getFirstChild();
                String  sMimeType = new String(aNode.getNodeValue());
                bMimeAccepted = aDataFacade.isMimeTypeAccepted( sSessionId, sMimeType);
            }
                // set accepted attribute
            if (bMimeAccepted)
            {
                aItemElement.setAttribute( "accepted", "true");
            }
            else
            {
                aItemElement.setAttribute( "accepted", "false");
            }
        }
    }
```

Fig. 11I

```
/*The global parameters for the XSL transformation are set and the loaded XML document is transformed according to the rules in the loaded
XSL stylesheet.
*/
                        aTransformerDoc = aDataFacade.setParameters(  aTransformerDoc, aRequest, aXSLParameters);
                        // transform
                        aTransformer.createTransform(aTransformerDoc)
                                                .transform(aSourceDoc,aDrainDoc);
/*The outputstream for the response is obtained and the transformed XML document is streamed, which now can be, for example, HTML or
WML to this stream.  The webserver delivers this stream as the response of the original request to the client.
*/
                        // write it
                        OutputStream aOutStream = aResponse.getOutputStream();
                        aResponse.setContentType(sPreferredMimeType);
                        aDrainDoc.write(aOutStream);
                        aOutStream.flush();
                }
        }
        catch (...
    }
}
```


Fig. 11J

Fig. 12